



**UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE TECNOLOGIA**

*Missão: Formar e aperfeiçoar cidadãos e prestar serviços atendendo às
necessidades tecnológicas da sociedade com agilidade, dinâmica e qualidade.*



PROJETO DE SO

Multicat Integer sort

Arthur Guedes RA: 213281

Kenzo Ishibashi RA:219630

Roberta Gomes RA:205443

Sumário

1 . Repositório no GITHUB	3
LINK GITHUB :	3
Link Vídeo:.....	3
2 . O problema – Solução proposta.....	3
3 . Instruções de compilação	4
4. Gráficos de tempo de Execução	5
Gráficos	5
Para somente Teste1.in na entrada e usando (2,4,8,16).....	5
Para somente Teste1.in e Teste2.in na entrada e usando (2,4,8,16).....	6
Para somente Teste1.in , Teste2.in e Teste3.in na entrada e usando (2,4,8,16)	6
5. Conclusão:.....	6

1 . Repositório no GITHUB

Segue o link para o repositório do GitHub com os seguintes itens:

- Relatório em PDF com link para download do vídeo
- Testes de entrada 1,2,3.
- Arquivo de Saida.
- Programa usado para criar os Testes 1,2,3.

Link Github:

<https://github.com/arthurghz01/so>

Link Vídeo:

<https://drive.google.com/file/d/1cfbm7siscPjK7zxedvKYle27DNqmCUh/view?usp=sharing>

2 . O problema – Solução proposta.

O projeto criado pela equipe Teletubbies visava criar um programa que utilizasse múltiplas threads (2,4,8,16) para ordenar uma série de números inteiros que o programa recebeu na entrada e posteriormente concatena-los e em seguida, realizar a análise do desempenho desse programa com 2,4,8 e 16 threads.

O desenvolvimento do programa adotado pelo grupo foi criar as funções separadamente e depois uni-las, declaração de variáveis globais e criação de uma STRUCT que serviu como base para passar os parâmetros para as threads.

O programa foi separado em níveis de dificuldades e ordem de necessidade, sendo criado a função que recebe todos os arquivos de entrada, realiza a leitura deles e escreve todos os dados em um arquivo de saída, ainda sem uma ordenação dos valores.

Em seguida, foi criada uma função para ler todos os dados do arquivo gerado anteriormente, essa função recebe todos os dados e os insere em um vetor alocado dinamicamente com as funções calloc e realloc, aumentando o vetor em 1000 unidades a cada vez que ele chegue próximo ao limite.

Após isso, veio o insight a respeito das threads, no momento que o usuário escolhe 2,4,8 ou 16 threads para operar, a função divide o valor para as threads, conforme o seguinte exemplo.

EX: O usuário inseriu 1000 dados e colocou para o programa executá-los em 2 threads, o programa cria 2 threads, a primeira ordena os dados do 0-500 e a segunda do 501-1000. Vale lembrar que a "sincronização" foi feita utilizando a função join, presente na biblioteca pré-requisito do projeto.

Vale a ressaltar que a ordenação é realizada com o algoritmo insertion sort, devido ele ter complexidade n^2 em todos os casos. E que o único tratamento de erro inserido no projeto é o caso do usuário inserir valores diferentes de 2,4,8 ou 16 para as threads. No caso do usuário inserir o nome de um arquivo que não existe, o programa irá mostrar a seguinte mensagem: "segment fault core dumped" e será encerrado.

Ao final de todo o processo descrito acima, foi obtido um programa que mostra-se eficiente para a - leitura, ordenação e escrita de dados concatenados – utilizando conceitos de múltiplas threads. Utilizando a biblioteca POSIX threads e sincronização

3 . Instruções de compilação

Realizar o download da Branch Master no github presente no repositório SO, em seguida, acessar via terminal a pasta no qual estão os arquivos baixados e inserir os seguinte comando.

gcc projetoSO.c -o projetoSO -lpthread

Após isso, basta apenas digitar:

Nome do executável - Numero de Threads – Arquivos.in - ArqSaida.

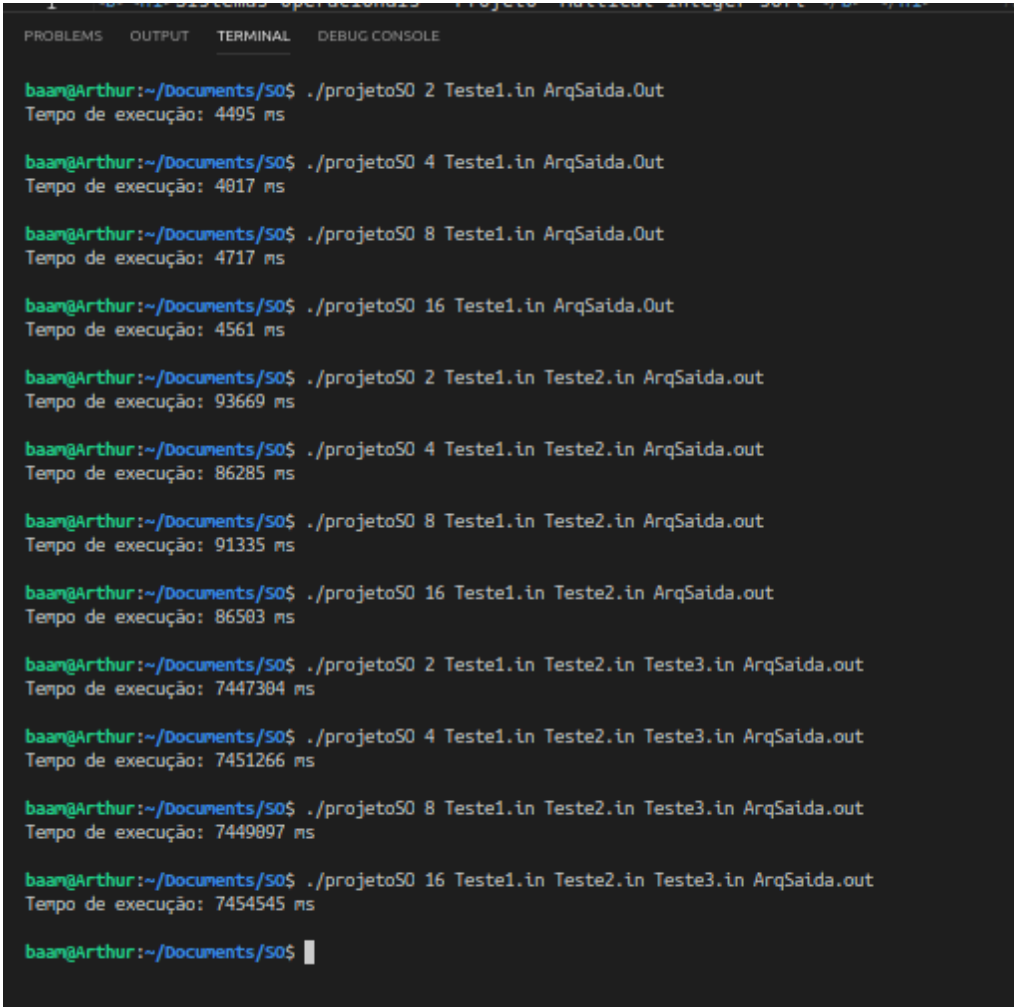
Exemplos:

./projetoSO 2 Teste1.in Teste2.in Teste3.in ArqSaida.out //PARA 2 THREADS

./projetoSO 4 Teste1.in Teste2.in Teste3.in ArqSaida.out //PARA 4 THREADS

./projetoSO 8 Teste1.in Teste2.in Teste3.in ArqSaida.out //PARA 8 THREADS

Segue abaixo foto de testes feito pelo grupo.



```
baam@Arthur:~/Documents/SO$ ./projetoSO 2 Teste1.in ArqSaida.Out
Tempo de execução: 4495 ms

baam@Arthur:~/Documents/SO$ ./projetoSO 4 Teste1.in ArqSaida.Out
Tempo de execução: 4017 ms

baam@Arthur:~/Documents/SO$ ./projetoSO 8 Teste1.in ArqSaida.Out
Tempo de execução: 4717 ms

baam@Arthur:~/Documents/SO$ ./projetoSO 16 Teste1.in ArqSaida.Out
Tempo de execução: 4561 ms

baam@Arthur:~/Documents/SO$ ./projetoSO 2 Teste1.in Teste2.in ArqSaida.out
Tempo de execução: 93669 ms

baam@Arthur:~/Documents/SO$ ./projetoSO 4 Teste1.in Teste2.in ArqSaida.out
Tempo de execução: 86285 ms

baam@Arthur:~/Documents/SO$ ./projetoSO 8 Teste1.in Teste2.in ArqSaida.out
Tempo de execução: 91335 ms

baam@Arthur:~/Documents/SO$ ./projetoSO 16 Teste1.in Teste2.in ArqSaida.out
Tempo de execução: 86503 ms

baam@Arthur:~/Documents/SO$ ./projetoSO 2 Teste1.in Teste2.in Teste3.in ArqSaida.out
Tempo de execução: 7447304 ms

baam@Arthur:~/Documents/SO$ ./projetoSO 4 Teste1.in Teste2.in Teste3.in ArqSaida.out
Tempo de execução: 7451266 ms

baam@Arthur:~/Documents/SO$ ./projetoSO 8 Teste1.in Teste2.in Teste3.in ArqSaida.out
Tempo de execução: 7449097 ms

baam@Arthur:~/Documents/SO$ ./projetoSO 16 Teste1.in Teste2.in Teste3.in ArqSaida.out
Tempo de execução: 7454545 ms

baam@Arthur:~/Documents/SO$
```

O programa irá mostrar apenas o tempo com processamento em MS de execução.

OS arquivos de entrada, possuem respectivamente: 1000, 10000 e 100000 mil números inteiros, tendo como valor mínimo de 0 intervalo de 10000.

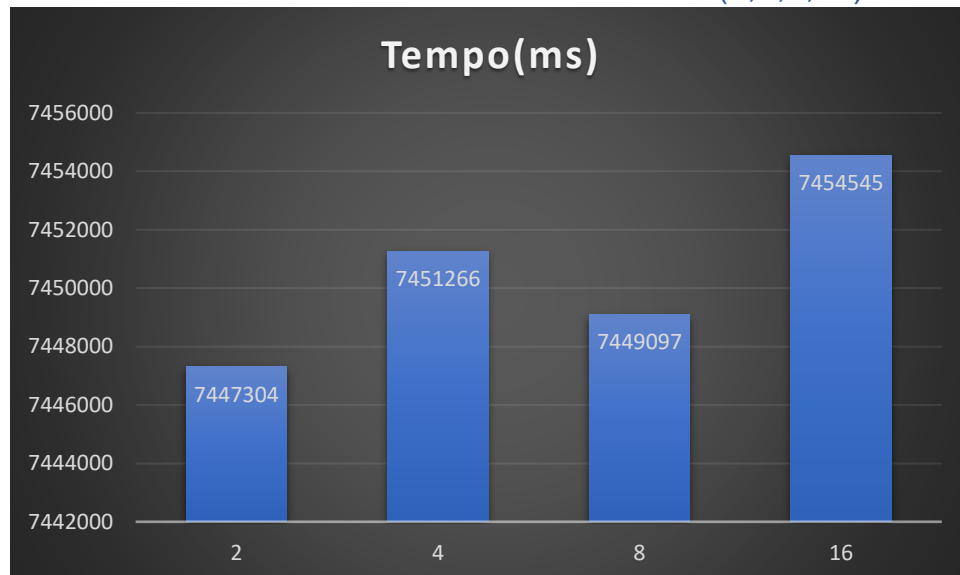
Sendo eles gerados por um programa feito pelo grupo que usa a função RAND da biblioteca STLIB

4. Gráficos de tempo de Execução

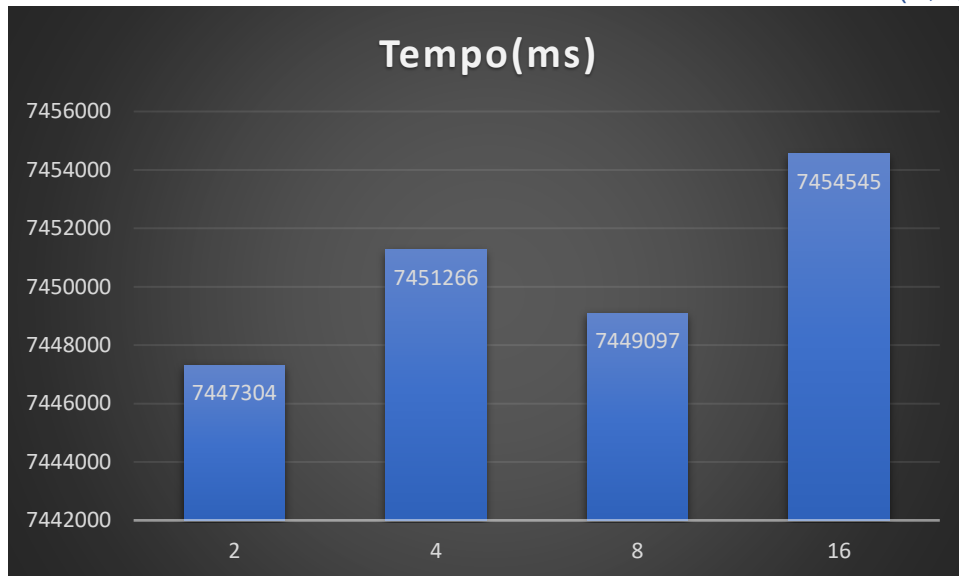
Ressalta-se que os testes foram realizados em um notebook **Lenovo 320 Intel Core i5 8250U 15,6" 8GB HD 1 TB GeForce MX150 e SO - UBUNTU 18.04.**

Gráficos

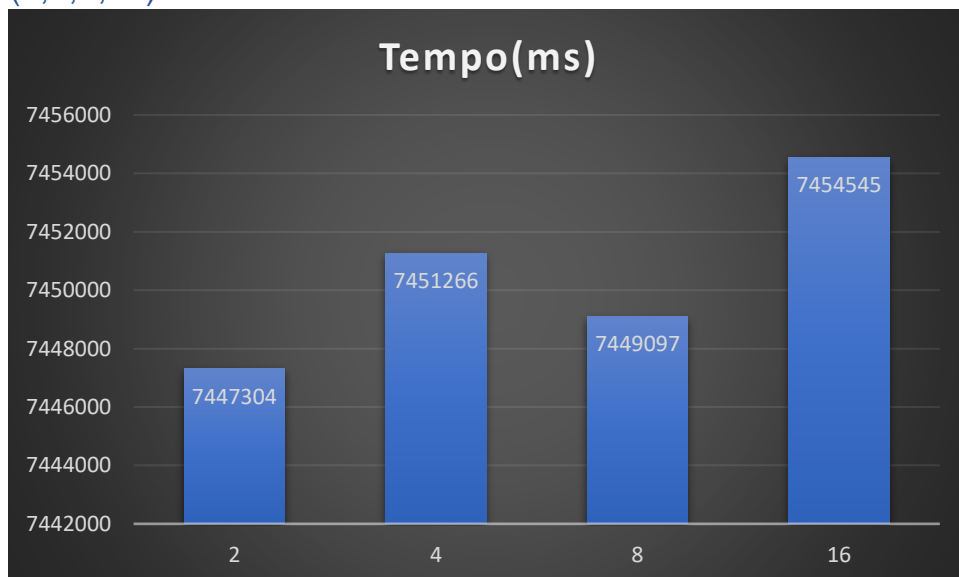
Para somente Teste1.in na entrada e usando (2,4,8,16)



Para somente Teste1.in e Teste2.in na entrada e usando (2,4,8,16)



Para somente Teste1.in , Teste2.in e Teste3.in na entrada e usando (2,4,8,16)



5. Conclusão:

Durante o processamento de um programa, ele é dividido em várias linhas, as quais possuem uma "ordem" de execução. Essas linhas tem processamento dividido pelo SO e repassadas ao processador. E basicamente assim que fundamenta-se o conceito de thread.

Com base nos testes feito pelo grupo, ao se executar um programa com uma grande variedade de dados é benéfico utilizar multiplas threads, e consegue se aumentar o desempenho conforme o número de threads vai aumentando. Por

outro lado, ao se executar pequenas quantias de dados utilizando conceitos de multi-threading você pode perder eficiência que teria ganho caso usasse apenas uma thread. Vale ressaltar que quantidade pequena ou grande de dados, é relativo conforme a capacidade de cada processador e definição.